```
// Computer Program Listing Appendix Under 37 CFR 1.52(e)
// ecachemgr.c
// Copyright (c) 2004, Sybase, Inc. All Rights Reserved.
/*
**  ECACHE_REPLACE_BUFFER
**
**  status = ecache_replace_buffer (mass_ptr)
**
** o grab a free secondary buffer
** o invokes ecache_copy_page which copies the page to secondary
**    cache (mmap operation)
** o In case of failure, put back the grabbed secondary buffer and
** return failure.
** o If operation successful, unhash from primary and hash it
**    secondary . Before doing that
**    check if it is KEPT (if a searcher found it before we could unhash
**    from primary). If true, unhash the buffer from secondary and the
**    copied page is destroyed and will be overwritten eventually.
**
**  During the secondary hashing, we are holding the primary
**  cache manager spinlock.
**
**  Parameters:
** buffer_ptr -- ptr to buffer to be replaced.
**
**  Returns:
** TRUE for success, FALSE otherwise
**
**  MP Synchronization:
** Caller has primary cache_spin lock.
** Acquires and releases secondary cache spin lock.
** Buffer must not be kept before being unhashed from primary.
**
**  Side Effects:
** buffer added to hash table or hash table overflow chain in
** secondary cache
** page copied to secondary cache (mmap operation)
** buffer unhashed from primary cache after checking that it is not kept.
**
**  History:
** 10/22/03 (rramani)  written
*/
SYB_BOOLEAN
ecache_replace_buffer (BUF * mass_ptr)
{
 SYB_BOOLEAN repl_in_sec; /* flag to replace in secondary
    ** cache.
    */
  ECACHE_BUF *sec_bp;  /* Secondary buffer */
 CACHE_DESC *primary_cdesc; /* primary cache descriptor */
```

```
ECACHE_DESC *sec_cdesc; /* secondary cache descriptor */
DES  *des;  /* DES of the object to which bp belongs*/
SYB_BOOLEAN unkeep_dbt; /* Indicate whether to unkeep dbt */
SYB_BOOLEAN unkeep_des; /* Indicate whether to unkeep des */
DBTABLE  *dbtable;
 ECACHE_BUF *hashed_bp;  /* Already Hashed Secondary buffer */
SYB_BOOLEAN ismassdirty; /* Is the mass marked dirty */
primary_cdesc = mass_ptr->bcache_desc;
des = NULL;
unkeep_dbt = FALSE;
unkeep_des = FALSE;
ismassdirty = FALSE;
SPINLOCKHELD(primary_cdesc->cspin);
/*
** If the buffer does not satisfy the required conditions, return
** FALSE. The buffer will follow the normal protocol of being
** washed from primary.
*/
if ((REPLACE_IN_ECACHE(mass_ptr) == FALSE) || IS_ECACHE_DISABLED)
{
 return FALSE;
}
/*
** MASS_WRITING will protect someone from
** reading a different page into this mass ie no
** one will be able to grab it. MASS_IO_NOT_STARTED also have to
** set to let others know that we are not really doing a disk
** write at this point.
*/
MASS_STAT(mass_ptr) |= (MASS_WRITING | MASS_IO_NOT_STARTED);
/* Note down if the mass is dirty */
ismassdirty = (MASS_STAT(mass_ptr) & MASS_DIRTY);
/*
** At this point we can release the spinlock because the buffer
** is under our control.
*/
V_SPINLOCK(primary_cdesc->cspin);
/*
** buf_hash_sec does the following:
** o grab a sec bp header bufgrab_sec();
** o acquire secondary cache descriptor, buffer pool = lowest
** o acquire secondary cache spinlock
** o copy required fields into sec_bp from bp
** o hash buffer in secondary hashtable
** o set MASS_READING in sec_bp;
** o release secondary spinlock
*/
/* Note thet we grab the buffer with out having to hold the
** spinlock. This is because ecache__grabmem() will take care of
** the wash synchronization and will return us a buffer that is
```

```
** unhashed and removed from the LRU/MRU chain.
*/
sec_bp =  ecache__grabmem();
/*
      ** If buffer grabbed is NULL and if extended cache is disabled,
      ** return FALSE.
      */
      if (!sec_bp)
      {
            P_SPINLOCK(primary_cdesc->cspin);
            MASS_STAT(mass_ptr) &= ~MASS_WRITING;
 if(!ismassdirty)
 {
  /*
  ** If the original mas was marked dirty then there
  ** may some clients who are sleeping on
  ** MASS_WRITING. But if we reset the
  ** MASS_IO_NO_STARTED bit then they will never
  ** issue the actual i/o. We will not clear this bit
  ** if the MASS is dirty
  */
  MASS_STAT(mass_ptr) &= ~MASS_IO_NOT_STARTED;
 }
            /* Wake up any task which is waiting for the i/o to complete */
            (void)upwakeup(SYB_EVENT_STRUCT(mass_ptr));
            return FALSE;
      }
sec_cdesc = Resource->recache_desc;
/*
** Copy page to secondary cache. The primary cache mgr spinlock
** is dropped. The secondary spinlock is already dropped at the end
** of ecach_bufhash. The spinlocks are dropped because of the
** mmap operation.
*/
repl_in_sec = ecache__copy_page (mass_ptr , sec_bp);
if(ismassdirty && repl_in_sec)
{
 dbtable = MASS_DBTABLE(mass_ptr);
 if( dbtable == NULL)
 {
  dbtable = dbt_get(MASS_DBID(mass_ptr), DBTF_CACHEONLY);
  if(dbtable)
  {
   unkeep_dbt = TRUE;
   MONITOR_INC(mc_ecache(ecache_replace_getdbt));
  }
  else
  {
   repl_in_sec = FALSE;
   P_SPINLOCK(primary_cdesc->cspin);
```

```
  P_SPINLOCK(sec_cdesc->ec_cache_spin);
  goto  finished;
 }
}
/*
** We need to fetch the DES of the object to which
** this buffer belongs. We need the DES to manipulate the the
** dirty chain and this is the only time we are not holding
** any spinlock. des_get needs spinlock and we need to this
** here.
*/
des = des_get(dbtable, INVALID_UID,
      mass_ptr->bpage->anp.pobjid, (BYTE *)NULL, 0,
      DES_ACTIVE_ONLY, FALSE);
MONITOR_INC(mc_ecache(ecache_replace_getdes));
/* If I couldn't get the des then mark replace a failure */
if (!des )
{
 unkeep_des = FALSE;
 repl_in_sec = FALSE;
}
else
{
 unkeep_des = TRUE;
}
}
/* Acquire both the spinlock */
P_SPINLOCK(primary_cdesc->cspin);
P_SPINLOCK(sec_cdesc->ec_cache_spin);
/*
** If the page is not copied to secondary cache, unhash it from
** secondary and return FALSE.
*/
if (!repl_in_sec)
{
 goto finished;
}
/*
** The buffer is hashed in secondary and the page has been
** copied successfully. Simply unhash it from primary if
** it is not kept. Also at this time if some one has unhashed
** from the buffer then just remove it from secondary as well.
*/
if ((MASS_STAT(mass_ptr) & MASS_KEPT) || !(MASS_STAT(mass_ptr) & MASS_HASHED))
{
 MONITOR_INC(mc_ecache(ecache_replace_foundkept));
 repl_in_sec = FALSE;
 goto finished;
}
/* Trasnfer the content of buffer header to secondary header */
```

```
 ecache__fill_buf(sec_bp, mass_ptr);
 /*
 ** Now is a good time fill the secondary buffer and hash it into
 ** the secondary.
 */
 if ((hashed_bp = ecache__bufhash(sec_bp)) != NULL)
 {
  /*
  ** This should never be the case as page can not be in
  ** both primary and secondary at the same time.
  */
  repl_in_sec = FALSE;
  V_SPINLOCK(sec_cdesc->ec_cache_spin);
  V_SPINLOCK(primary_cdesc->cspin);
  ex_raise(BUFFERM, DUP_BHASH, EX_CMDFATAL, 3,
    MASS_LPAGENO(mass_ptr),
    MASS_DBID(mass_ptr),
    sizeof(EXTENDED_CACHE_NAME),
    EXTENDED_CACHE_NAME);
 }
finished:
 if (repl_in_sec)
 {
  /*
  ** We need to erase the identity of primary buffer from the
  ** primary cache. Transfer the remaning part of the primary
  ** buffer to the extended buffer to complete the copy operation.
  */
  ecache__rm_pbuf(sec_bp, mass_ptr, des);
  SYB_ASSERT(!(MASS_STAT(mass_ptr) & MASS_DIRTY));
  /* Insert the replaced buffer in the MRU end */
  INSQHEAD(&sec_cdesc->ec_bufhead, sec_bp);
 }
 else
 {
  /*
  ** For some reason replacement failed. Put the buffer in
  ** the LRU end and decrement the wash deficit
  */
  ECACHE_BUF_STAT(sec_bp) |= MASS_INWASH;
  sec_cdesc->ec_bwashdeficit --;
  INSQTAIL(&sec_cdesc->ec_bufhead, sec_bp);
  ecache__clear_buf(sec_bp);
  MONITOR_INC(mc_buffer(mass_ptr->bcache_desc, buf_copy_page_2k_fail));
 }
 /* Now denote that the copy operation is completed. */
 ECACHE_BUF_STAT(sec_bp) &= ~MASS_BEING_COPIED;
 V_SPINLOCK(sec_cdesc->ec_cache_spin);
 /* If we had kept the des unkeep it now */
 if( unkeep_des || unkeep_dbt )
```

```
{
 V_SPINLOCK(primary_cdesc->cspin);
 if(unkeep_des)
 {
  des_unkeep(des);
 }
 /* Also unkeep the dbt if we had kept it */
 if (unkeep_dbt)
 {
  dbt_unkeep(dbtable);
 }
 P_SPINLOCK(primary_cdesc->cspin);
}
/*
** Now that we have removed the primary buffer reset MASS_WRITING
** bit.
*/
MASS_STAT(mass_ptr) &= ~MASS_WRITING;
if(!ismassdirty)
{
 /*
 ** If the original mas was marked dirty then there
 ** may some clients who are sleeping on
 ** MASS_WRITING. But if we reset the
 ** MASS_IO_NO_STARTED bit then they will never
 ** issue the actual i/o. We will not clear this bit
 ** if the MASS is dirty
 */
 MASS_STAT(mass_ptr) &= ~MASS_IO_NOT_STARTED;
}
/* Wake up any task which is waiting for the i/o to complete */
(void)upwakeup(SYB_EVENT_STRUCT(mass_ptr));
if (repl_in_sec)
{
 MONITOR_INC(mc_buffer(mass_ptr->bcache_desc, buf_replace_sec_2k));
 MONITOR_INC(mc_ecache(ecache_replaced));
}
return repl_in_sec;
}
/*
**  ECACHE_OBTAIN_BUFFER (bp, sdes)
**
**  status = ecache_obtain_buffer (bp, sdes)
**
** o primary spinlock is not held. But the buffer has been hashed into
**   primary hashtable and has MASS_READING set (in bufread)
** o Take the secondary cache spinlock.
** o sec_bp = ecache__bufsearch(bp->bdbid, bp->bpageno).
** o Not found in secondary
**  - release secondary spinlock.
```

```
**  - return FALSE.
** o Found in secondary:
**    o It cannot be in the process of writing or write cannot
**      start unless we finish the copy.
**      So unlink from secondary lru-mru chain.
**    o Release secondary spinlock.
**    o Invoke ecache_copy_page with target=bp->bpage and source=
**      sec_bp->offset.
**    o If copy fails, mmap fails. Raise a disk i/o exception and
**      hang.
**    o if copy succeeds:
**     - Check if any des flush is active and
**    sec_bp->eb_dirtyseq <
**    sec_bp->eb_dbtable->dbt_nextseq
**    - TRUE => write it out.
**     - Transfer reqd info from secondary to primary buffer
**     hdr
**    - status bits
**    - dirty chain
**     - Use the eb_dbtable pointer to dbt
**     - Invoke des_get to get the des.
**     - Get the first dirty chain element.
**       Take the pin sequence number of it
**       and assign 1 less to this bp
**       (using bufdlink or something)
**   - Make sure no one looks at anything in buffer when
**     it has MASS_READING bit set.
**   - 'bp' is setup correctly now.
**   - Take the secondary spinlock and unhash from secondary
**     This is just a safety net.
**   - 'sec_bp' should have its status cleared and linked
**       at the LRU end of secondary cache.
**   - The page in secondary is now free for others to use
**     because of the DESTROYED status in 'sec_bp'.
**   - Now that all operations are done, reset MASS_READING
**     in primary bp.
**
**  Parameters:
** bp - ptr to buffer to be populated.
**
**  Returns:
** TRUE for success, FALSE otherwise
**
**  MP Synchronization:
** Caller has set MASS_READING in primary buffer header (after hashing it)
** Acquires and releases secondary cache spin lock.
** The mmap operation from secondary cache must not crib.
**
**  Side Effects:
**
```

```
** secondary cache spinlock obtained and released.
** if page is found in secondary, sec_bp could be unhashed and replaced
** at LRU end.
** page copied to primary bp (mmap operation)
** in case of failure during copy, hang similar to disk i/o error.
**
**  History:
** 11/26/03 (rramani)  written
*/
SYB_BOOLEAN
ecache_obtain_buffer (BUF * bp, SDES *sdes)
{
 SYB_BOOLEAN got_from_sec; /* flag to set from secondary
    ** cache.
    */
  ECACHE_BUF *sec_bp;  /* Secondary buffer */
 ECACHE_DESC *sec_cdesc; /* secondary cache descriptor */
 circ_long flush_seq; /* the sequence number that is used
    ** to decide whether the buffer needs
    ** to be written before read from
    ** secondary. */
 BUF  *mass_ptr; /* Mass pointer for given buffer */
 CACHE_DESC *primary_cdesc; /* primary cache descriptor */
 DES  *des;  /* Pointer to the DES of the buffer
    ** being copied
    */
 DBTABLE  *dbtable; /* DBTABLE of the buffer we are copying
    ** incase we need to get the DES for
    ** the same.
    */
 SYB_BOOLEAN unkeep_des; /* Flag to indicate whether to
    ** unkeep the DES
    */
 SYB_BOOLEAN unkeep_dbt; /* Flag to indicate whether to
    ** unkeepo the DBTABLE
    */
 /*
 ** If the buffer does not satisfy the required conditions, return
 ** FALSE. It will be read from disk. This check is to see if at all
 ** this buffer could have been a candidate earlier for secondary
 ** replacement.
 */
 if ((FOUND_IN_ECACHE(bp) == FALSE))
  return FALSE;
 got_from_sec = TRUE;
 unkeep_des = FALSE;
 unkeep_dbt = FALSE;
 des = NULL;
 SYB_ASSERT(bp->bmass_stat & MASS_READING);
 sec_cdesc = Resource->recache_desc;
```

```
 mass_ptr = bp->bmass_head;
 primary_cdesc = bp->bcache_desc;
 MONITOR_INC(mc_ecache(ecache_srchcalls));
get_spin:
 /*
 ** Acquire the secondary spinlock to start the search.
 */
 P_SPINLOCK(sec_cdesc->ec_cache_spin);
 sec_bp = ecache__bufsearch(bp->bdbid, bp->blpageno);
 /* Not found in secondary cache */
 if (sec_bp == NULL)
 {
  V_SPINLOCK(sec_cdesc->ec_cache_spin);
  return FALSE;
 }
 /*
 ** Check if the sec_bp is in the process of being written. Sleep
 ** and check again.
 */
 if (sec_bp->eb_status & MASS_WRITING)
 {
  V_SPINLOCK(sec_cdesc->ec_cache_spin);
  MONITOR_INC(mc_ecache(ecache_read_writewait));
  (void) upsleepgeneric(SYB_EVENT_NON_STRUCT(&sec_bp->eb_status),
    (char *) &sec_bp->eb_status,
    sizeof(sec_bp->eb_status),
    (long) MASS_WRITING,
    ((TRUE << 8) | FALSE),
    MDANAP_ECACHE_1);
  goto get_spin;
 }
 /* Mark that we are reading from this secondary buffer */
 ECACHE_BUF_STAT(sec_bp) |= MASS_READING;
 /*
 ** If the page that we are looking was found in the wash region
 ** increment the wash deficit and reset the status as we are
 ** removing it from here. Eventhough eventually we will put this
 ** buffer back into the wash region, between now and the time we
 ** put back many threads may have gotten buffers from the queue
 ** that may eventually cause deficit calculation to be inconsistent.
 */
 if (ECACHE_BUF_STAT(sec_bp) & MASS_INWASH)
 {
  ECACHE_BUF_STAT(sec_bp) &= ~MASS_INWASH;
  sec_cdesc->ec_bwashdeficit++;
  /*
  ** If the buffer that we are removing is the one which
  ** is the wash marker then we need set the wash marker to
  ** the next buffer.
  */
```

```
 if(sec_bp == sec_cdesc->ec_washmarker)
 {
  sec_cdesc->ec_washmarker =
   (ECACHE_BUF *)QUE_NEXT(&sec_bp->eb_buf_link);
  SYB_ASSERT(sec_cdesc->ec_washmarker !=
    (ECACHE_BUF *)&sec_cdesc->ec_bufhead);
 }
}
/*
** Just unlink sec_bp from the LRU MRU chain. This way it is
** protected and there is no need for the spinlock.
*/
REMQUE(sec_bp, sec_bp, ECACHE_BUF);
/* Release the spinlock */
V_SPINLOCK(sec_cdesc->ec_cache_spin);
/*
** Copy page from secondary cache. The primary cache mgr spinlock
** is dropped. The secondary spinlock is already dropped at the end
** of ecach_bufhash. The spinlocks are dropped because of the
** mmap operation.
*/
got_from_sec = ecache__get_page (bp, sec_bp);
/*
** If the page is not got from secondary cache, we are doomed. Just
** hang as if it were an I/O error.
*/
if (got_from_sec == FALSE)
{
  if (sdes)
  {
  bufunkeep(bp->bmass_head, sdes, UNLATCH_IF_LATCHED);
  }
            ex_raise(BUFFERM, B_HARDERR, EX_HARDWARE, 4,
                        PH_HARDREAD, bp->bmass_head);
}
/*
** If the mass is dirty then we need the handle on the DES of the
** mass that we are transering. If the caller had supplied a valid
** SDES and we can  get to the DES from there. Howerver if SDES is
** NULL then we have to fetch the DES. Note that DES can't be NULL
** the MASS to which it belongs is still DIRTY.
*/
if((ECACHE_BUF_STAT(sec_bp) & MASS_DIRTY) &&
   (!sdes || ( sdes->sdesp->dobjectc.objostat.objid != sec_bp->eb_objid)))
{
 /*
 ** At this time we would have filled the valid dbtable in
 ** the bp.
 */
 dbtable = MASS_DBTABLE(bp);
```

```
if(  dbtable == NULL)
{
 dbtable = dbt_get(bp->bdbid, DBTF_CACHEONLY);
 unkeep_dbt = TRUE;
}
/*
** We need to fetch the DES of the object to which
** this buffer belongs. We need the DES to manipulate the the
** dirty chain and this is the only time we are not holding
** any spinlock. des_get needs spinlock and we need to this
** here. Note that copy is done. We do have a valid page at
** this time.
*/
des = des_get(dbtable, INVALID_UID,
          bp->bpage->anp.pobjid, (BYTE *)NULL, 0,
      DES_ACTIVE_ONLY, FALSE);
/* If I couldn't get the des then mark replace a failure */
SYB_ASSERT(des);
unkeep_des = TRUE;
}
else if((ECACHE_BUF_STAT(sec_bp) & MASS_DIRTY))
{
 des = sdes->sdesp;
}
/*
** Also assert that the primary buffer and secondary buffer headers have
** exact informatio.
*/
SYB_ASSERT(sec_bp->eb_dbid == MASS_DBID(bp));
SYB_ASSERT(sec_bp->eb_lpageno == MASS_LPAGENO(bp));
 SYB_ASSERT(bp->bpage->anp.pobjid == sec_bp->eb_objid);
/*
** The page has been copied successfully. Destroy its identity in
** secondary cache. Also also a part of this function, do the
** necessary transfer between the primary and secondary buffer
** headers.
**  o Changing from secondary dirty chain to primary dirty chain will
**    be done only when the buffer is dirty.
**  o At this also check if any des flush operation if happening.
**  o Assign pin sequence number = pin sequence of first guy - 1
**  o secondary buffer header is unhashed, attached to LRU end.
**  o address having the page will be re-used.
** We need to hold the primary spinlock because while trasnfering
** the content of the secondary to primary buffer we may have to
** insert the mass to the dirty chain. For that we need to hold
** the primary spinlock. One optimization is to hold the spinlock
** only if the secondary is dirty.
*/
P_SPINLOCK(primary_cdesc->cspin);
P_SPINLOCK(sec_cdesc->ec_cache_spin);
```

```
 ecache__rm_sbuf(sec_bp, bp, des);
 /* We are done reading from the secondary buffer */
 ECACHE_BUF_STAT(sec_bp) &= ~MASS_READING;
 V_SPINLOCK(sec_cdesc->ec_cache_spin);
 /* Reset MASS_READING to denote that the read is completed */
 bp->bmass_stat &= ~MASS_READING;
 V_SPINLOCK(primary_cdesc->cspin);
 /* If we had kept the des unkeep it now */
 if( unkeep_des || unkeep_dbt )
 {
  if(unkeep_des)
  {
   des_unkeep(des);
  }
  /* Also unkeep the dbt if we had kept it */
  if (unkeep_dbt)
  {
   dbt_unkeep(dbtable);
  }
 }
 /* We may need to wakeup thread waiting for read to complete */
 (void)upwakeup(SYB_EVENT_STRUCT(mass_ptr));
 MONITOR_INC(mc_ecache(ecache_read));
 /*
 **  We are going to decrement the counter as it will be incremented
 ** immediately following bufread. This decrement will offset the
 ** increment done in get_pagewith_validation. However some time
 ** after the bufread this is not incremented. So incrementing here
 ** we may cause the counter to go negative.
 */
 if (sdes)
 {
  sdes->sbufread--;
 }
 return TRUE;
}
/*
** ECACHE__BUFSEARCH
**
**  This routines searches for (dbid, page) in the secondary hashtable.
**
** Parameters
**
** dbid -- databse id
** pageno  -- page number
**
** Returns
**  Pointer to secondary buffer header if found, NULL otherwise.
**
** MP Synchronization:
```

```
** secondary cachemgr spinlock is held.
**
** History
**  Written  - 11/27/03 (rramani)
*/
ECACHE_BUF *
ecache__bufsearch(dbid_t dbid, pgid_t pageno)
{
 ECACHE_DESC  *edesc;  /* Local handle to extended cache desc*/
 ECACHE_BUF **hashbucket; /*
     ** Hash bucket to which the given
    ** buffer was hashed.
    */
 ECACHE_BUF *tmpbuf; /* Temporary buffer handle */
 edesc = Resource->recache_desc;
 /*
 ** Make sure that the caller has the spinlock for the extended
 ** cache.
 */
 SPINLOCKHELD(edesc->ec_cache_spin);
 /* Get the hash bucket entry */
 hashbucket = ECACHE_BUF_HASH(edesc, dbid, pageno);
 if (*hashbucket)
 {
  tmpbuf = *hashbucket;
  /* Locate if buffer exist in the overflow chain */
  for(tmpbuf = *hashbucket; tmpbuf;
      tmpbuf = tmpbuf->eb_hashtab_link )
  {
   if((tmpbuf->eb_dbid == dbid) &&
      (tmpbuf->eb_lpageno == pageno))
   {
    /* Found a match */
    return(tmpbuf);
   }
  }
 }
 /*
 ** Didn't find the entry. Read from disk.
 */
 return (NULL);
}
```